CMPT 317 Assigment 1

Chin Wang Lau(chl929), Qishan Mao(qim853), Hao Li(hal215)

12 February 2016

1 Problem Description

In this assignment, we are given a map with M locations, N vehicles each with P capability and K packages. The map is a graph with M locations, include the garage. Each package should have a source and a destination, which should be two different locations. All vehicles have the same capability P. The vehicles should start at garage and return to the same garage after they finished delivery all the packages. Our goal is to find the smallest sum of the path length for all vehicles.

2 Solution Description

Concretely speaking, this problem is a mixture of VRPPD (Vehicle Routing Problem with Pickup and Delivery) and CVRP (Capacitated Vehicle Routing Problem), or a specialization of the VRP (Vehicle Routing Problem). What important is the nature: this problem is a NP-hard (Non-deterministic Polynomial hard) problem that no one has found an effective search algorithm which can get a surely optimal solution when the size of input data is large. In another word, due to the low efficiency, using exact search algorithm that always finds the optimal solution is unrealistic when having a considerable number of vehicles and/or packages.

However, with a trade-off in a little of the solution quality, approximation algorithm can find a pleasant but not optimal solution within general time limits. Since the solution quality and the time cost may have dramatic difference with different sorts of approximation algorithm, we choose to use genetic search algorithm as our main search algorithm because of its excellent balance in the two requirements.

The method of our solution is simple: use a-star path search algorithm to compute the best paths between any pair of locations and convert the map into graph; use genetic algorithm to search the best set of package arrangement and breakdown the problem into TSP (Traveling Salesman Problem); use a greedy algorithm, whose heuristic is to try the closest valuable location first, to find a good path and the related path length for each vehicle and evaluate the efficiency of each set of package arrangement.

For further development, our solution can be redesigned and calibrated in order to achieve higher quality of solution, lower time cost, ability to solve other variations of VRP. For higher performance, we can replace the arrangement evaluation function with Taboo Search or exact algorithm, replace the package arrangement function with another genetic algorithm or Simulate Anneal Arithmetic. The VRP variations include“ last package in, first package out ” , time window, allowing a vehicle to travel multiple trips, allowing a vehicle not to return to the garage , length limit in the route, and etc.

3 Implementation Description

We choose to use Python as the programming language. The reasons are: codes of random graph is provided; Python can invoke a-star path search through importing networkx; Python is excellent in group programming. The explanations of out functions is sorted with the process order and listed below.

The first four steps are converting the descripted problem into the structured data that can be solved by the machine.

The first step our program does is to create random map without specific locations implemented by the file “Map.py”. The function, “makeMap()”, takes the dimension of the map and the gap frequency as parameters. In the end of the function, a random Cartesian coordinate map with weight will be created.

The second step is to create a random location for the garage. The function setGarageLocation() is used for garage location and takes one parameter which is the map created in the first step. After this function gets end, it creates a random location for garage on the map.

The third step is to create random packages whose source and destination are in different locations. There are two functions setPickupLocation() and setDeliverLocation(). These functions take two parameters which are the number of packages and the map created above. Each package has two fields: source of the package and destinations of the package. After these functions get end, the list of pick up locations and the list of deliver locations will be returned, and the each pair locations should be different.

The fourth step is to convert the map into an undirected graph or table constituted by vertices and weighted edges. Since all vehicles will only be travelling between locations, we only need to know all distance between any pair of the locations. Without saving the paths between pair of the locations, memory will be saved. With recording the distance, a part of time will be saved from calculating the distance again and again. The function findShortestPathGraph() takes the map as parameter created by the first step. The distance between locations is gathered by tracing the a-star path search algorithm. After this function gets end, the map of the question will be simplified into the undirected graph with weighted edges.

Then we can start the genetic algorithm. The first step of genetic algorithm is to initialize the population, which is the initial solution. Each solution here is represented as an array whose length is the number of the packages. For a

solution, an element are an integer which is the serial number of the vehicle that responsible for delivering this package. This coding a solution like an array is based on the following assumptions: without requirement in the path length nor time, a vehicle can delivery any number of packages; if the packages delivered by a vehicle and the method of path evaluation are known, the optimal path of a vehicle can be found while only one optimal path of this vehicle has to be found. In our algorithm, the initial solution, or array, is gained through greedy algorithm whose method is closest valuable location first. Then, we create the whole initial population by mutating the copies of the initial solution. After, we start the loop through the following steps:

Evaluation:
use another greedy algorithm, which is very similar as the previous one, to calculate the total path length of each solution and sort them with the calculated result. Compare the best solution in the population with the saved best solution. If the saved best solution is worse or null, save the best solution in the population as the saved best solution.

Termination:
check if the number of passed loop has reach the goal loop number. If the number has been reach, terminate the loop and return the saved best solution; else, increase the number of passed loop by 1 and continue the loop.

Selection:
according to the result of sort, select the best half minus 1 of the solutions and the saved best solution.

Crossover:
create the new generation to restore the initial population size by repeating the loop: randomly select a pair of solutions; randomly create a crossover point (from 0 to the size of survived solutions) for each pair; create a new solution by combining the part before crossover point in the first solution of the pair and the part not before crossover point in the second solution of the pair.

Mutation:
For each new-born solution, mutate each of their elements with given mutation possibility. Then, jump to the step 1, Evaluation. ...

The rest function is to use the greedy algorithm mentioned in the Evaluation to trace and output the path of each vehicles. For checking the correctness of the whole algorithm, all the created map, graph, locations, and packages will be outputted.

4 Result

Example Graph and result:

The graph limited as a 5 x 5 graph with gapfreq → 0.45. To show more simple and easy understand on our work and result.

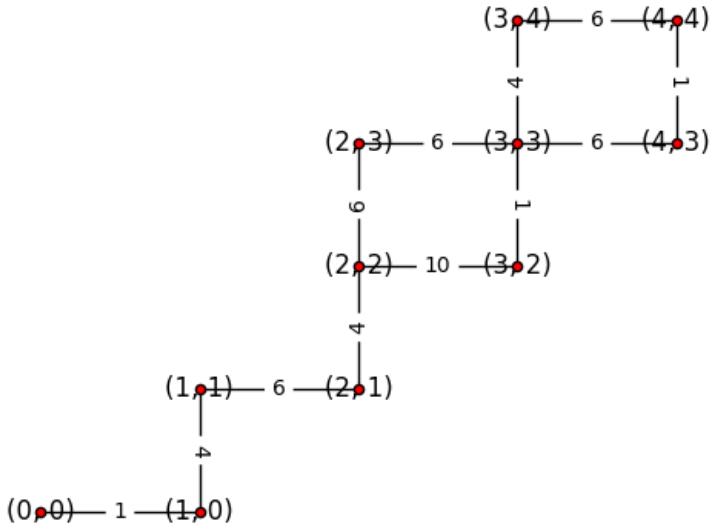

Figure 1: This is the node that exist on our graph

[(2, 1), (3, 3), (4, 4), (2, 2), (1, 1), (3, 2), (0, 0), (2, 3), (1, 0), (4, 3), (3, 4)]

Here is the graph which get result form a star algorithm that measure the minimum weight from each node to another from a* algorithm.

Node	(2, 1)	(3, 3)	(4, 4)	(2, 2)	(1, 1)	(3, 2)	(0, 0)	(2, 3)	(1, 0)	(4, 3)	(3, 4)
(2, 1)	0	15	22	4	6	14	11	10	10	21	19
(3, 3)	15	0	7	11	21	1	26	6	25	6	4
(4, 4)	22	7	0	18	28	8	33	13	32	1	6
(2, 2)	4	11	18	0	10	10	15	6	14	17	15
(1, 1)	6	21	28	10	0	20	5	16	4	27	25
(3, 2)	14	1	8	10	20	0	25	7	24	7	5
(0, 0)	11	26	33	15	5	25	0	21	1	32	30
(2, 3)	10	6	13	6	16	7	21	0	20	12	10
(1, 0)	10	25	32	14	4	24	1	20	0	31	29
(4, 3)	21	6	1	17	27	7	32	12	31	0	7
(3, 4)	19	4	6	15	25	5	30	10	29	7	0

Total number of node in the map: 11

Garage location: (2, 2)

Package number: 2

Deliver location list: [(2, 1), (0, 0)]

Pick up location list: [(3, 3), (2, 1)]

This mean pick up location (3,3) → deliver location (2,1) and pick up location (2, 1) → deliver location (0, 0)

Since the package is not created as a Class, there is an error when it is combined with the genetic algorithm which makes it difficult to approach to the final result.

5 Conclusion

We were trying to using python and genetic algorithm to solve the searching problem. Since we made "packages" as a list instead of a class, accessing the destination after picking up a package became too difficult. The inexperience to python and genetic algorithm was also a big trouble for us. Although we still learned a lot about python and AI search algorithm, our implementation got failed in searching for a possible solution. However, our algorithm design and documentation are satisfying.

To improve our work, we should learn more about the management of development to improve the efficiency of out team work, so that the "package" problem may not happened. Then the idea of algorithm can also be better if we designed it more carefully. Also we should gain more experience with python and data structure